计算机组成

CPU形式建模综合方法
——单周期数据通路

高小鹏

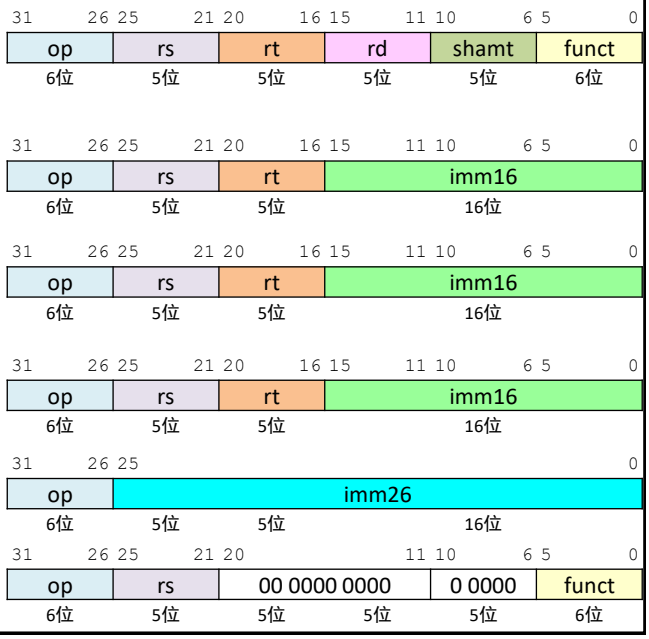
北京航空航天大学计算机学院

目录

- 设计方法学概述
- 指令级别数据通路与控制器建模
- 数据通路综合方法

MIPS-C0指令集

- 加减法指令
  - `addu rd,rs,rt`
  - `subu rd,rs,rt`
- 立即数指令
  - `ori rt,rs,imm16`
- 存储访问指令
  - `lw rt,rs,imm16`
  - `sw rt,rs,imm16`
- 分支指令
  - `beq rs,rt,imm16`
- 函数调用指令
  - `jal target`
- 函数返回指令
  - `jr rs`



MIPS-C0指令集

- 从功能角度，MIPS-C0可以构造程序设计的绝大多数功能
  - lw、sw：存储指令的典型代表
  - addu、subu：运算类指令的典型代表
  - beq：分支类指令的典型代表
  - jal、jr：支持函数
- 从结构角度，MIPS-C0包含了所有3种指令格式
  - R型：addu、subu、jr
  - I型：lw、sw、beq
  - J型：jal

注意
Jr是R型指令，而不是J型指令

MIPS-C0的RTL描述

- 任何指令执行的第一步都是取指令

  R-format: {op, rs, rt, rd, shamt, funct} ← MEM[PC]

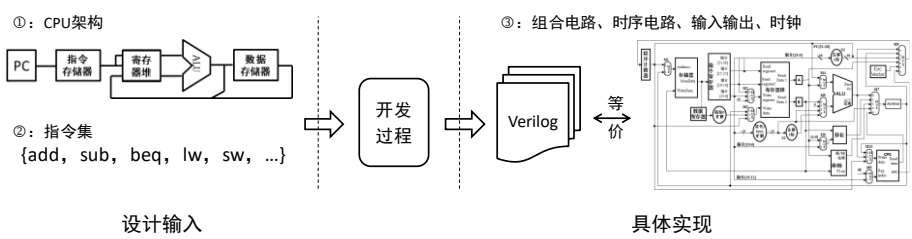
  I-format: {op, rs, rt, imm16} ← MEM[PC]

- MIPS-C0的RTL描述

指令	RTL描述
addu	R[rd]←R[rs]+R[rt]; PC←PC+4
subu	R[rd]←R[rs]-R[rt]; PC←PC+4
ori	R[rt]←R[rs]|zero_ext(imm16); PC←PC+4
lw	R[rt]←MEM[R[rs]+sign_ext(imm16)]; PC←PC+4
sw	MEM[R[rs]+sign_ext(imm16)]←R[rt]; PC←PC+4
beq	if (R[rs] == R[rt]) then PC←PC+4 + (sign_ext(imm16) || 00) else PC←PC+4
jal	PC←PC31..28||instr_index||02; R[31]←PC + 4
jr	PC←R[rs]

形式建模综合方法概述

- CPU开发是什么？
  - 以某种CPU架构（如单周期、多周期或流水线）为模型，面向给定的指令集，设计并构造出以Verilog形式表达的CPU具体实现



- CPU架构模型是什么？
  - CPU架构模型：以寄存器堆、ALU等功能部件为基础，描述了各功能部件的基本接口特性以及相互间的基本连接关系
  - CPU架构模型关注两点
    - 功能部件的外特性，即功能部件的某个具体功能与控制的对应关系
    - 功能部件间传递信息的基本依赖关系

形式建模综合方法概述

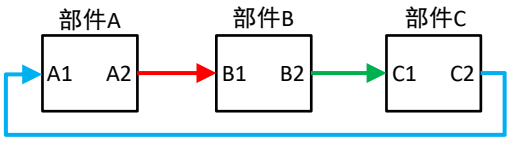
- 形式建模综合方法的要点是什么？
  - 1）开发过程是基于模型的
    - 这里的模型就是所谓的CPU架构，如单周期、多周期、流水线
  - 2）开发过程被显式的分为设计和实现两个环节
    - 设计：是指建模每条指令的数据通路和控制信号
    - 实现：是指将设计结果用Verilog等语言表达
  - 3）基于“系统-子系统”视角的建模层次
    - CPU被视为系统，各功能部件被视为子系统
  - 4）指令级别独立建模
    - 独立分析每条指令操作语义，推演其执行过程中的数据流信息和控制流信息
    - 独立构造每条指令对应的数据通路以及相关功能部件的控制信号取值
  - 5）一次性的系统级综合
    - 将所有分离的数据通路及控制信号取值高效合成为完整的数据通路和控制器

形式建模综合方法概述

- 指令级别独立建模是什么？
  - 1）根据指令的RTL，梳理和总结出数据通路的设计需求
  - 2）选择恰当的数据通路功能部件
  - 3）建立功能部件间的正确连接关系
  - 4）根据指令的RTL，选择功能部件应执行的功能，反推控制信号取值

建模的基本表示方式

- 数据通路本质上是一个连接关系的集合
  - 集合的元素：相关部件的输入输出信号之间的连接关系
- 示例：3个部件的连接关系



部件	输入	输出
A	A1	A2
B	B1	B2
C	C1	C2

  - 连接关系集合：{<C.C2,A.A1>，<A.A2,B.B1>，<B.B2,C.C1>}
    - 连接关系<X.m,Y.n>：部件X的输出信号m，连接至部件Y的输入信号n
  - 连接关系表格：该方式表示连接关系集合，在布局上更易于对应图形方式

A	B	C	------ 部件
A1	B1	C1	------ 部件的输入信号
C.C2	A.A2	B.B2	------ 输入来源：部件及其输出信号

建模的基本推理过程

- 基本事实1：当CPU基本模型确定后，意味着指令的执行路径总体是确定的
  - 执行路径也可以认为是信息流路径
  - 执行路径大体包括：取指、译码/读操作数、计算、访存、回写
- 基本事实2：构成指令执行路径的各分段之间存在着明确的依赖关系
  - 示例1：指令执行的前提是必须先读取指令；而读取指令就必然是用PC驱动IM的地址线，然后IM输出的就是当前要执行的指令
  - 示例2：读取DM的前提是必须先计算出地址；而地址是通过加法运算完成的，因此ALU执行加法就是非常合理的

建模的推理方法：倒推法

- 由于指令执行存在强逻辑依赖关系，因此建模指令基本的数据通路时既可采用正向推理方法，也可采用反向推理法，即倒推法
  - 反向推理结束后，将推理顺序倒置即得到正向的推理顺序

环节	步骤	语义	连接关系	功能部件	控制信号取值
-1	A	每步的具体操作	<X.m,Y.n>	F	信号名:取值
-2	B				
-3	C				

环节	步骤	语义	连接关系	功能部件	控制信号取值
1	C				
2	B				
3	A	每步的具体操作	<X.m,Y.n>	F	信号名:取值

数据通路表

- 为了直观展示及易于综合，每条指令推理结束后，用数据通路表记录该指令的所有连接关系
- 示例：包含PC、NPC、IM、RF、ALU、DM的数据通路表
  - 第1行：为功能部件名
  - 第2行：该功能部件的各个输入信号
  - 第3行：与该输入信号连接的某个功能部件的输出信号

指令	NPC	PC	IM	RF				ALU		DM	
	PC	DI	A	A1	A2	A3	WD	A	B	A	WD

目录

- 设计方法学概述
- 指令级别数据通路与控制器建模
- 数据通路综合方法

Addu建模

R[rd]←R[rs]+R[rt]

- 最后环节：必定是将加法运算的计算结果写入寄存器堆
- 写入寄存器堆涉及3个要素，即写入编号、写入数据、写使能
  - 基本事实：ALU具有加法功能
  - 推理1：写入编号来自指令的rd域
  - 推理2：写入数据来自ALU
  - 推理3：写使能需要有效，即为1

31–26	25–21	20–16	15–11	10–6	5–0
special 000000	rs	rt	rd	0 00000	addu 100001
6	5	5	5	5	6

环节	步骤	语义	连接关系	功能部件	控制信号取值
-1	回写	计算结果回写至rd寄存器	<ALU.C,RF.WD> <IM.D[15:11],RF.A3>	RF	RFWr:1

Addu建模 R[rd]←R[rs]+R[rt]

- 将连接关系用表格表示
  - 控制信号取值暂时不表示，后续会在控制器设计中使用

环节	步骤	语义	连接关系	功能部件	控制信号取值
-1	回写	计算结果回写至rd寄存器	<ALU.C,RF.WD> <IM.D[15:11],RF.A3>	RF	RFWr:1

指令	NPC	PC	IM	RF				ALU		DM	
				A1	A2	A3	WD	A	B	A	WD
addu						IM.D[15:11]	ALU.C				

Addu建模 R[rd]←R[rs]+R[rt]

- -2环节：既然写入数据来自ALU，因此需要由ALU完成加法计算
- ALU运算涉及3个要素，即第1个操作数、第2个操作数、ALU操作
  - 推理1：第1个操作数是从RF的RD1输出
  - 推理2：第2个操作数是从RF的RD2输出
  - 推理3：ALU控制信号取值为加法编码

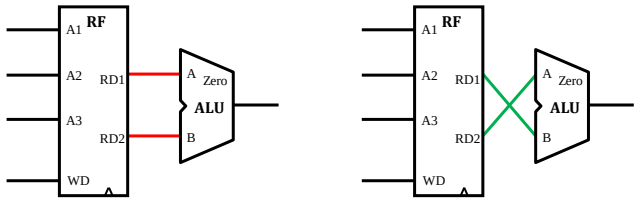
这是具有可读性的宏定义

环节	步骤	语义	连接关系	功能部件	控制信号取值
-1	回写	计算结果回写至rd寄存器	<ALU.C,RF.WD> <IM.D[15:11],RF.A3>	RF	RFWr:1
-2	执行	ALU执行加法	<RF.RD1,ALU.A> <RF.RD2,ALU.B>	ALU	ALUOp:ADD

指令	NPC	PC	IM	RF				ALU		DM	
	PC	NPC	A	A1	A2	A3	WD	A	B	A	WD
addu				IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C	RF.RD1	RF.RD2		

RF与ALU的连接关系

- RF有2个32位输出，RD1和RD2；ALU有2个输入，A和B
- 问题：所有指令的RTL都不定义两者的连接关系，那么ALU的A/B连接RF的RD1/RD2还是RD2/RD1？



- 解答
  - 1）对于任一指令而言，两种连接均可以
  - 2）推荐采用固定连接方案（习惯上采用方案A）

※※※
如果有些指令采用方案A，有些指令采用方案B，这不影响功能正确性。但是，今后综合完整数据通路时需要更多MUX。

Addu建模

R[rd]←R[rs]+R[rt]

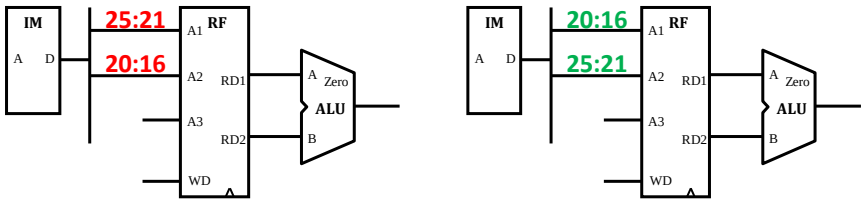
- -3环节：为了支持ALU计算，RF需要读取2个操作数
- RF读取操作数涉及2个要素，即第1操作数编号、第2操作数编号
  - 推理1：第1操作数编号是指令的rs域（先假设RF的RD1输出对应rs）
  - 推理2：第2操作数编号是指令的rt域（先假设RF的RD2输出对应rt）

环节	步骤	语义	连接关系	功能部件	控制信号取值
31 26	25 21	20 16	15 11	10 6	5 0
special 000000	rs	rt	rd	0 00000	addu 100001
6	5	5	5	5	6
-3	读操作数		<IM.D[25:21],RF.A1> <IM.D[20:16],RF.A2>	RF	

指令	NPC	PC	IM	RF				ALU		DM	
	PC	NPC	A	A1	A2	A3	WD	A	B	A	WD
addu				IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C	RF.RD1	RF.RD2		

IM与RF的连接关系

- RF有2个读寄存器编号，A1和A2
- 问题：RF的A1/A2对应**rs/rt**还是**rt/rs**？



A)　　B)

- 解答
  - 1）对于任一指令而言，两种连接均可以
  - 2）推荐采用固定连接方案（为保持一致性，习惯上采用方案A）

31–26	25–21	20–16	15–11	10–6	5–0
special 000000	rs	rt	rd	0 00000	addu 100001
6	5	5	5	5	6

Addu建模

R[rd]←R[rs]+R[rt]

- -4环节：必然是取指令
- 取指令涉及2个要素，即读取IM、计算PC+4、更新PC
  - 推理1：为了读取IM，需要PC驱动IM的地址线
  - 推理2：为了计算PC+4，需要PC输入至NPC
  - 推理3：为了更新PC，需要将NPC计算结果再写入PC

环节	步骤	语义	连接关系	功能部件	控制信号取值
-1	回写	计算结果回写至rd寄存器	<ALU.C,RF.WD> <IM.D[15:11],RF.A3>	RF	RFWr:1
-2	执行	ALU执行加法	<RF.RD1,ALU.A> <RF.RD2,ALU.B>	ALU	ALUOp:ADD
-3	读操作数		<IM.D[25:21],RF.A1> <IM.D[20:16],RF.A2>	RF	
-4	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <NPC.NPC,PC.DI>	IM PC NPC	

指令	NPC	PC	IM	RF				ALU		DM	
	PC	DI	A	A1	A2	A3	WD	A	B	A	WD
addu	PC.DO	NPC.NPC	PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C	RF.RD1	RF.RD2		

Subu建模

R[rd]←R[rs]-R[rt]

- Subu与addu高度相似，区别仅仅在于ALU执行减法运算

环节	步骤	语义	连接关系	功能部件	控制信号取值
1	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <NPC.NPC,PC.DI>	IM PC NPC	
2	读操作数		<IM.D[25:21],RF.A1> <IM.D[20:16],RF.A2>	RF	
3	执行	ALU执行减法	<RF.RD1,ALU.A> <RF.RD2,ALU.B>	ALU	ALUOp:**SUB**
4	回写	计算结果回写至rd寄存器	<ALU.C,RF.WD> <IM.D[15:11],RF.A3>	RF	RFWr:1

指令	NPC	PC	IM	RF				ALU		DM	
	PC	DI	A	A1	A2	A3	WD	A	B	A	WD
subu	PC.DO	NPC.NPC	PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C	RF.RD1	RF.RD2		

Ori建模

R[rt]←R[rs] OR zero_ext(imm16)

- Ori与addu/sub非常相似，区别在于
  - 1）ALU的第2个操作数为EXT的扩展结果
  - 2）ALU执行OR运算
  - 3）写入rt寄存器

环节	步骤	语义	连接关系	功能部件	控制信号取值
1	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <NPC.NPC,PC.DI>	IM PC NPC	
2	读操作数		<IM.D[25:21],RF.A1> <IM.D[15:00],EXT.IMM16>	RF EXT	
3	执行	ALU执行逻辑或	<RF.RD1,ALU.A> <EXT.Ext,ALU.B>	ALU	ALUOp:**OR**
4	回写	计算结果回写至rd寄存器	<ALU.C,RF.WD> <IM.D[20:16],RF.A3>	RF	RFWr:1

指令	NPC	PC	IM	RF				EXT	ALU		DM	
	PC	NPC	A	A1	A2	A3	WD	Imm16	A	B	A	WD
addiu	PC.DO	NPC.NPC	PC.DO	IM.D[25:21]		IM.D[20:16]	ALU.C	IM.D[15:0]	RF.RD1	EXT.Ext		

Lw建模

R[rt]←MEM[R[rs]+sign_ext(imm16)]

- ALU输入寄存器与扩展数，执行加法，结果输出至DM地址
  - 1）地址计算与ori的计算过程类似（寄存器与扩展数运算）
  - 2）ALU具有执行加法的功能
- 为完成回写，DM的数据输出连接至RF的WD，且DM写使能为1

环节	步骤	语义	连接关系	功能部件	控制信号取值
1	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <NPC.NPC,PC.DI>	IM PC NPC	
2	读操作数		<IM.D[25:21],RF.A1> <IM.D[15:00],EXT.IMM16>	RF EXT	EXTOp:SEXT
3	执行	ALU执行加法	<RF.RD1,ALU.A> <EXT.Ext,ALU.B>	ALU	ALUOp:ADD
4	访存	读取DM	<ALU.C,DM.A>	DM	
4	回写	计算结果回写至rd寄存器	<DM.RD,RF.WD> <IM.D[20:16],RF.A3>	RF	RFWr:1

31 26	25 21	20 16	15 0
lw 100011	base	rt	offset

Lw建模

R[rt]←MEM[R[rs]+sign_ext(imm16)]

环节	步骤	语义	连接关系	功能部件	控制信号取值
1	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <NPC.NPC,PC.DI>	IM PC NPC	
2	读操作数		<IM.D[25:21],RF.A1> <IM.D[15:00],EXT.IMM16>	RF EXT	EXTOp:SEXT
3	执行	ALU执行加法	<RF.RD1,ALU.A> <EXT.Ext,ALU.B>	ALU	ALUOp:ADD
4	访存	读取DM	<ALU.C,DM.A>	DM	
5	回写	计算结果回写至rd寄存器	<DM.RD,RF.WD> <IM.D[20:16],RF.A3>	RF	RFWr:1

指令	NPC	PC	IM	RF				EXT	ALU		DM	
	PC	NPC	A	A1	A2	A3	WD	Imm16	A	B	A	WD
lw	PC.PC	NPC.NPC	PC.PC	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[15:0]	RF.RD1	EXT.Ext	ALU.C	

- 注意：由于EXT部件被修改了，因此凡是与该部件相关的指令均需被重新建模
  - 主要是建模需要考虑EXT的控制信号取值

Sw建模

MEM[R[rs]+sign_ext(imm16)]←R[rt]

- Sw与lw在地址计算方面完全一致
- Sw与lw不同点在于其不需要读DM，而是将rt寄存器写入DM
  - 即RF的RD2要写入DM

环节	步骤	语义	连接关系	功能部件	控制信号取值
1	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <NPC.NPC,PC.DI>	IM PC NPC	
2	读操作数		<IM.D[25:21],RF.A1> <IM.D[15:00],EXT.IMM16>	RF EXT	EXTOp:SEXT
3	执行	ALU执行加法	<RF.RD1,ALU.A> <EXT.Ext,ALU.B>	ALU	ALUOp:ADD
4	访存	写入DM	<ALU.C,DM.A> <RF.RD2,DM.WD>	DM	DMWR:1

指令	NPC	PC	IM	RF				EXT	ALU		DM	
	PC	NPC	A	A1	A2	A3	WD	Imm16	A	B	A	WD
lw	PC.PC	NPC.NPC	PC.PC	IM.D[25:21]				IM.D[15:0]	RF.RD1	EXT.Ext	ALU.C	RF.RD2

Beq建模

```
if ( R[rs] == R[rt] )
    PC←PC+4 + sign_ext(imm16||00)
else
    PC←PC+4
```

- NPC计算PC
- ALU执行减法

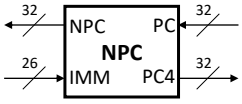
环节	步骤	语义	连接关系	功能部件	控制信号取值
1	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <IM.D[15:0],NPC.Imm> <NPC.NPC,PC.DI>	IM PC NPC	Br:1 Zero:ALU的zero
2	读操作数		<IM.D[25:21],RF.A1> <IM.D[20:16],RF.A2 >	RF	
3	执行	ALU执行减法	<RF.RD1,ALU.A> <RF.RD2,ALU.B>	ALU	ALUOp:SUB

指令	NPC		PC	IM	RF				ALU		DM	
	PC	Imm	NPC	A	A1	A2	A3	WD	A	B	A	WD
beq	PC.PC	IM.D[15:0]	NPC.NPC	PC.PC	IM.D[25:21]	IM.D[20:16]			RF.RD1	RF.RD2		

Jal建模

$PC \leftarrow PC_{31..28} \| instr_index \| 0^2$
$R[31] \leftarrow PC+4$

- 包含2个操作：①计算PC值；②将PC+4写入R31
- NPC改造1：支持新的PC计算需求
  - NPC的功能定位是计算PC值
  - NPC已输入指令低16位，只需要扩大至指令低26位，就能满足全部需求
  - NPC的计算需求包含3种情况，分别是PC+4、PC+4+偏移、jal地址；必须将br调整为NPCOp[1:0]
    - br信号只有1位，不能表达3种含义
- NPC改造2：支持回写
  - NPC已经计算出了PC+4，因此只需要将PC+4输出即可



※图中未包含控制类信号

31 26	25 0
jal 000011	Instr_index

Jal建模

$PC \leftarrow PC_{31..28} \| instr_index \| 0^2$
$R[31] \leftarrow PC+4$

- 调整后的NPC的I/O信号

信号名	方向	描述	调整说明
PC[31:0]	I	32位输入	
Imm[25:0]	I	26位立即数	从Imm[15:0]调整为Imm[25:0]
NPCOp[1:0]	I	NPC功能选择 00：计算顺序地址（PC+4） 01：计算beq地址 10：计算jal地址 11：保留	取消1位的Br，改为2位的NPCOp
Zero	I	rs和rt相等标志 1：相等 0：不等	
NPC[31:0]	O	32位输出	
PC4[31:0]	O	32位输出	输出PC+4值

注意：由于NPC的控制信号发生了变化，因此前面各条指令的NPC控制信号取值也需要随之调整。

Jal建模

$PC \leftarrow PC_{31..28} \| instr_index \| 0^2$
R[31]←PC+4

- 对于回写数据来说，数据来自NPC输出的PC4
- 对于回写寄存器编号来说，需要直接表示为0x1F
  - 其并未出现在指令格式中，而是固定为31

环节	步骤	语义	连接关系	功能部件	控制信号取值
1	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <IM.D[25:0],NPC.Imm> <NPC.NPC,PC.DI>	IM PC NPC	NPCOp:JAL
2	回写		<0x1F,RF.A3> <NPC.PC4,RF.WD >	RF	RFWr:1

指令	NPC		PC	IM	RF				EXT	ALU		DM	
	PC	Imm	DI	A	A1	A2	A3	WD	Imm	A	B	A	WD
jal	PC.DO	IM.D[25:0]	NPC.NPC	PC.DO			0x1F	NPC.PC4					

目录

- 设计方法学概述
- 指令级别数据通路与控制器建模
  - Jr
- 数据通路综合方法
- 控制器综合方法
- 单周期CPU性能分析

Jr建模

PC←R[rs]

- 设计分歧：虽然Jr的功能很简单，但却有2种可能思路
  - 方案1：将RF输出的rs值直接输出至PC
  - 方案2：将RF输出的rs值先输出至NPC，然后再从NPC输出至PC
- 基本思路：遵循某些基本原则或方法
- 基本原则：“高内聚低耦合”中的最少知识原则
  - 方案1：PC与NPC都是“知道”如何计算PC值的功能部件
  - 方案2：NPC是唯一“知道” 如何计算PC值的功能部件

31 26	25 21	20 11	10 6	5 0
special 000000	rs	0 00 0000 0000	0 00000	jr 001000

Jr建模

PC←R[rs]

- NPC接口：①增加函数返回地址RA[31:0]；②增加新的计算模式

信号名	方向	描述	调整说明
PC[31:0]	I	32位输入	
Imm[25:0]	I	26位立即数	从Imm[15:0]调整为Imm[25:0]
RA[31:0]	I	32位返回地址	rs寄存器保存的返回地址
NPCOp[1:0]	I	NPC功能选择 00：计算顺序地址（PC+4） 01：计算beq地址 10：计算jal地址 11：计算jr地址	NPCOp的0b11项用于产生jr相关的PC目的地址
Zero	I	rs和rt相等标志 1：相等 0：不等	
NPC[31:0]	O	32位输出	
PC4[31:0]	O	32位输出	输出PC+4值

注意：由于NPC的控制信号发生了变化，因此前面各条指令的NPC控制信号取值也需要随之调整。

Jr建模

PC←R[rs]

- 推理发现存在冲突：同一个部件执行多个操作。如何选择？
  - 在环节1和3，NPC执行不同的计算功能

环节	步骤	语义	连接关系	功能部件	控制信号取值
1	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <NPC.NPC,PC.DI>	IM PC NPC	NPCOp:PC4 ①
2	读操作数		<IM.D[26:21],RF.A1>	RF	
3	计算返回地址		<RF.RD1,NPC.RA>	NPC	NPCOp:JR ②
4	更新PC	将rs值写入PC	<NPC.NPC,PC.DI>	PC	

- 选择后执行的，即环节3。原因：
  - 单周期特点是所有操作都必须在一个时钟周期内完成
  - 逻辑上，后发生的事件是会覆盖先发生的事件的

```
// a的值为6
a = 5 ;
a = 6 ;
```


Jr建模

PC←R[rs]

- 单周期的特点是所有操作都必须在一个时钟周期内完成。这意味

环节	步骤	语义	连接关系	功能部件	控制信号取值
1	读指令	读取指令 计算下条指令地址 更新PC	<PC.DO,IM.A> <PC.DO,NPC.PC> <NPC.NPC,PC.DI>	IM PC NPC	~~NPCOp:PC4 ①~~
2	读操作数		<IM.D[26:21],RF.A1>	RF	
3	计算返回地址		<RF.RD1,NPC.RA>	NPC	NPCOp:JR ②
4	更新PC	将rs值写入PC	<NPC.NPC,PC.DI>	PC	

指令	NPC			PC	IM	RF				EXT	ALU		DM	
	PC	Imm	RA	DI	A	A1	A2	A3	WD	Imm	A	B	A	WD
jal	PC.DO		RF.RD1	NPC.NPC	PC.DO	IM.D[25:21]								

目录

- 设计方法学概述
- 指令级别数据通路与控制器建模
- 数据通路综合方法

汇聚

- 将独立建模的指令级别数据通路汇聚在一起

部件	PC	NPC			IM	RF				EXT	ALU		DM
输入信号	DI	PC	Imm	RA	A	A1	A2	A3	WD		A	B	A
addu	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2	
subu	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2	
ori	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	ALU.C	IM.D[15:0]	RF.RD1	EXT.Ext	
lw	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[15:0]	RF.RD1	EXT.Ext	ALU.C
sw	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]		IM.D[15:0]	RF.RD1	EXT.Ext	ALU.C
beq	NPC.NPC	PC.DO	IM.D[15:0]		PC.DO	IM.D[25:21]	IM.D[20:16]				RF.RD1	RF.RD2	
jal	NPC.NPC	PC.DO	IM.D[25:0]	RF.RD1	PC.DO			0x1F	NPC.PC4				
jr	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]							

综合

- 归并每个输入信号的信号来源：保留不同来源

部件	PC	NPC			IM	RF				EXT	ALU		DM	
输入信号	DI	PC	Imm	RA	A	A1	A2	A3	WD		A	B	A	D
addu	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2		
subu	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11]	ALU.C		RF.RD1	RF.RD2		
ori	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	ALU.C	IM.D[15:0]	RF.RD1	EXT.Ext		
lw	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]	DM.RD	IM.D[15:0]	RF.RD1	EXT.Ext	ALU.C	
sw	NPC.NPC	PC.DO			PC.DO	IM.D[25:21]		IM.D[20:16]		IM.D[15:0]	RF.RD1	EXT.Ext		RF.RD2
beq	NPC.NPC	PC.DO	IM.D[15:0]		PC.DO	IM.D[25:21]	IM.D[20:16]				RF.RD1	RF.RD2		
jal	NPC.NPC	PC.DO	IM.D[25:0]	RF.RD1	PC.DO			0x1F	NPC.PC4					
jr	NPC.NPC	PC.DO		RF.RD1	PC.DO	IM.D[25:21]								
完整	NPC.NPC	PC.DO	IM.D[25:0] ※	RF.RD1	PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11] IM.D[20:16] 0x1F	ALU.C DM.RD NPC.PC4	IM.D[15:0]	RF.RD1	RF.RD2 EXT.Ext	ALU.C	

※由于IM.D[25:0]覆盖了IM.D[15:0]，因此只保留前者即可

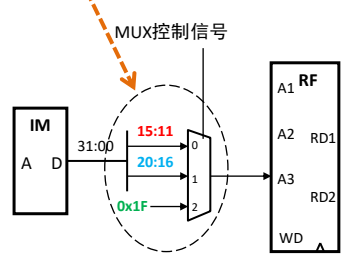
计算机组成与实现

构造MUX

- 以RF.A3为例，综合后的结果表明该信号有3个输入源：
  - IM.D[15:11]、IM.D[20:16]、0x1F

部件	PC	NPC			IM	RF				EXT	ALU		DM
输入信号	DI	PC	Imm	RA	A	A1	A2	A3	WD		A	B	A
完整	NPC.NPC	PC.DO	IM.D[25:0]	RF.RD1	PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11] IM.D[20:16] 0x1F	ALU.C DM.RD NPC.PC4	IM.D[15:0]	RF.RD1	RF.RD2 EXT.Ext	ALU.C

- 根据数字电路知识可知，必须部署一个MUX
- 由此可见，MUX是“自动”综合出来



※在控制部分再详述MUX的工程化规范设计

计算机组成与实现

构造MUX

- MUX各路输入信号分派并无特定的原则
  - 示例：RF的A3信号

指令与RF.A3

指令	RF.A3
addu	IM.D[15:11]
subu	IM.D[15:11]
ori	IM.D[20:16]
lw	IM.D[20:16]
sw	IM.D[20:16]
beq	
jal	0x1F
jr	

RF.A3的MUX

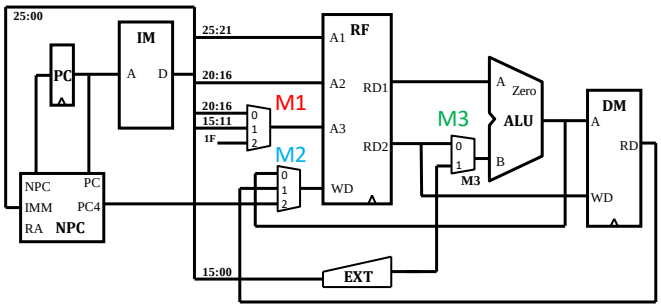
端口编号	输入源
0	IM.D[15:11]
1	IM.D[20:16]
2	0x1F

注意
一旦确定输入源与MUX端口的对应关系后，务必在数据通路、控制信号等设计上确保一致性

数据通路的两种表示方式

- 从形式建模的数据通路可以很容易的构造出图形化的数据通路

部件	PC	NPC			IM	RF				EXT	ALU		DM
输入信号	DI	PC	Imm	RA	A	A1	A2	A3	WD		A	B	A
完整	NPC.NPC	PC.DO	IM.D[25:0]	RF.RD1	PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11] IM.D[20:16] 0x1F	ALU.C DM.RD NPC.PC4	IM.D[15:0]	RF.RD1	RF.RD2 EXT.Ext	ALU.C
								M1	M2			M3	



从设计模型到VerilogHDL

- 示例：ALU的B输入

部件	PC	NPC			IM	RF				EXT	ALU		DM
输入信号	DI	PC	Imm	RA	A	A1	A2	A3	WD	Imm	A	B	A
完整	NPC.NPC	PC.DO	IM.D[25:0]	RF.RD1	PC.DO	IM.D[25:21]	IM.D[20:16]	IM.D[15:11] IM.D[20:16] 0x1F	ALU.C DM.RD NPC.PC4	IM.D[15:0]	RF.RD1	RF.RD2 EXT.Ext	ALU.C

```
wire     [31:0]     RD2 ;
wire     [31:0]     Ext ;
wire     [31:0]     ALU_B ;

RF     U_RF( ..., RD2, ... ) ;          // 实例化寄存器堆
EXT    U_EXT( ..., Ext, ... ) ;         // 实例化扩展单元
ALU    U_ALU( ..., ALU_B, ... ) ;       // 实例化ALU

// 实例化
MUX32_2_1 U_MUX_ALUB( RD2, Ext, ALU_B, ALUBSrc ) ;
```